A Novel Multiple Key Block Ciphering Mechanism with Reduced Computational Overhead

V.S.Shankar Sriram
Dept. Of Information Technology
Birls Institute of Technology
Mesra,Ranchi,India-835215

Abhishek Kumar Maurya
Dept. Of Computer Science & Engg.
Birla Institute of Technology
Mesra,Ranchi,India-835215

G.Sahoo
Dept. Of Information Technology
Birla Institute of Technology
Mesra,Ranchi,India-835215

ABSTRACT
Cryptanalysis of symmetric key cryptography encourages large key size and complex operations to achieve message confidentiality. All these techniques pose computational overhead at both the sender & receiver ends. In this paper, we propose a simple yet powerful Block Cipher Multiple Key Symmetric Encryption (BCMKSE) algorithm for achieving both confidentiality & integrity with reduced computational and message overheads. Our algorithm changes the key after encrypting/decrypting a piece of the whole message. While the key changes during the whole message encryption/decryption process without increasing network traffic or message overhead. This methodology becomes faster as it uses the simplest operations like shift, XOR, addition and comparison operations.

Categories and Subject Descriptors

D.4.6 [**Security and Protection**]: Access controls, Authentication, Cryptographic controls, Information flow controls

General Terms

Security

Keywords

Block Cipher, Confidentiality, Cryptography, Message Integrity, Symmetric Key.

1. INTRODUCTION
Symmetric-key algorithms are a class of algorithms for cryptography that use trivially related, often identical, cryptographic keys for both decryption and encryption. The encryption key is trivially related to the decryption key, in that they may be identical or there is a simple transform to go between the two keys. The keys, in practice, represent a shared secret between two or more parties that can be used to maintain a private information link. Other terms for symmetric-key encryption are secret-key, single-key, shared-key, one-key and eventually private-key encryption [1].

There are two ways in which the plaintext is processed in the cryptography: block cipher and stream cipher. The block cipher method divides a large data set into blocks (based on predefined size or the key size), processes the input block of elements one at a time, producing an output block for each input block. A stream cipher processes the element continuously, producing one output element at a time, as it goes along [2].

All the symmetric key cryptographic algorithms encourage using large key size and/or complex procedure and/or multiple keys (but limited number of keys i.e. two, three, four, ten, etc.) in the ciphering process to achieve confidentiality. Examples of such algorithms are Data Encryption Standard (DES), Advance Encryption Standard

All the symmetric key cryptographic algorithms encourage using large key size and/or complex procedure and/or multiple keys (but limited number of keys i.e. two, three, four, ten, etc.) in the ciphering process to achieve confidentiality. Examples of such algorithms are Data Encryption Standard (DES), Advance Encryption Standard (AES), and Blowfish etc. The mechanisms used by these algorithms follow complex procedure for encryption and decryption. Some like Triple DES (3DES) follows complex procedure and multiple keys technique to make it difficult for an attacker to decipher it. These mechanisms not only increase the complexity for the attacker to crack, but also puts a lot of computational overhead to the sender and receiver in the process of ciphering and deciphering. Employing Complex encryption procedure to secure messages at the cost of speed is not a preferred one. Using multiple keys in the ciphering process is a good alternative. But it puts other types of overhead in terms of the key generation and exchange process which results in increased network traffic. Why do not we have such a mechanism which doesn't use any complex process that, can reduce the network traffic, can also achieve message integrity and does not encourage using large key size? With this in mind we have designed a Block Cipher Multiple Key Symmetric Encryption (BCMKSE) algorithm for achieving both confidentiality & integrity with reduced computational overhead.

Rest of this paper is organized as follows. Section 2 addresses the computational overhead in some popular existing symmetric key algorithms. Section 3 explores our proposed work (i.e. Key Generation and Block Cipher Multiple Key Symmetric Encryption (BCMKSE) algorithms). Section 4 will describe its benefits. Simulation results are discussed in section 5 followed by conclusion in section 6.

2. OVERHEADS IN EXISTING SYMMETRIC KEY ALGORITHMS

It is worth to mention here that the existing symmetric key encryption algorithms pose several computational overheads. A brief note on the operational mechanism and overheads posed by some popular symmetric key algorithms are discussed below.

2.1 DES/3DES

DES applies a symmetric 56-bit key to each 64-bit block of data. The process can run in several modes and involves 16 rounds of operations. DES is breakable as the key size is too less. Hence Triple DES (3DES) an Enhancement of DES emerged as a stronger method. Triple DES encrypts the data three times and uses a different key for at least one of the three passes for giving a cumulative key of size 112-168 bits. If we consider a triple length key to consist of three 56-bit keys K1, K2, K3 then encryption is as following order :- encryption with K1, decryption with K2, encryption with K3. Whereas, decryption is the reverse process of encryption, so the decryption of the cipher text will be as follows: - decryption with K3, encryption with K2, decryption with K1 [3].

The computational overhead at the sender and receiver ends, are as high as 3DES that involves three times the operations of normal DES, with each DES contributing 16 rounds of operation [4]. Further the increasing key size increases the process run time complexity. Moreover the keys are of fixed size and do not change with sessions, they has to be changed periodically between the parties for achieving more message confidentiality.

2.2 AES

An Advanced Encryption Standard (AES) has basically three different configurations with respect to the number of rounds and key sizes. It has 10 rounds for 128-bit keys, 12 rounds for 192-bit keys, and 14 rounds for 256-bit keys. By 2006, the best known attacks were on 7 rounds for 128-bit keys, 8 rounds for 192-bit keys, and 9 rounds for 256-bit keys [5, 6]. It is worth mentioning here that any increase in the number of iterations or rounds and key size is a burden for both the sender and the receiver. Even the key is fixed and it doesn't change with the respective session. So, using a single key for long time decreases the message confidentiality.

2.3 Blowfish Encryption

Blowfish encrypts data in 8-byte blocks. The algorithm consists of two parts: a key-expansion part and a data-encryption part. Key expansion converts a variable-length key of at most 56 bytes (448 bits) into several sub key arrays totaling 4168 bytes. Blowfish has 16 rounds. Each round consists of a key dependent permutation, and a key and data-dependent substitution. All operations are XORs and additions on 32-bit words [7, 8]. Blowfish Encryption is modified version of DES with simplified operations like XORs and additions. The run time complexity of Blowfish is much lesser than that of DES and AES, using the same concept, but suffers the same problem as DES.

Session key uses multiple keys for ciphering and deciphering the message for a period of time. In session key mechanism after a particular time t, both the sender and receiver will change the keys which are being used to cipher and decipher. The problem with this technique is that if there is lack of synchronization between the sender and receiver, one end might have changed the key whereas other might be using the previous key. But instead of using a time stamp to change a key, we consider here the change of key after encrypting some number of bits (i.e. some piece of message) and similarly on receiver side it changes the key after decrypting same number of bits. We emphasize here that this novel idea is more suitable for this process of encryption or decryption which has been justified through experimental results.

3. BLOCK CIPHER MULTIPLE KEY SYMMETRIC ENCRYPTION (BCMKSE) ALGORITHM

On the basis of the above mentioned problems of the popular crypto algorithms, we propose Block Cipher Multiple Key Symmetric Encryption (BCMKSE) algorithm which doesn't suffer from any of the overheads mentioned above. Our algorithm falls under the category of symmetric key block ciphering and uses a key of 128 bits. The numbers of keys which are used to cipher/decipher are dependent on message size and the NOB (number of bits) variable. NOB variable will decide, after how many number of bits the key should change, where the decimal value of NOB has been taken from random number and the number of bits required to represent NOB variable, are agreed previously by both communicating parties. The proposed algorithm is based upon the following consideration.

We consider 128 bits key size and n bits NOB. The decimal value of n-bit NOB is random in nature where 7<=n<=128. The NOB variable will decide, after how many bits of encryption/decryption, the key should change. This NOB variable must be greater than zero and in the multiples of 128. The key and NOB will be known to both the sender and receiver by the exchange of an initial message. The multiple numbers of keys will depend on the overall message size and NOB. At the sender's end, key will change after transmitting NOB bit of original message and at the receiver's end, after receiving same number of bits, key will be changed. For message integrity, we will use message authentication code (MAC) using MD5.

The encryption and decryption process of the proposed algorithm uses bit_count as a variable to count the number of bits to be encrypted and decrypted. The bit_count will be compared to NOB. If it is equal to the NOB, a new key will be generated. The encryption and decryption will then be performed with the new key generated; otherwise the message will be encrypted or decrypted by the same key.

3.1 Key Generation Algorithm

Let us assume that there is a server S and there are i number of nodes in the network where each node is represented by N_i. The key and NOB will be generated at the server end and will be exchanged between the server and the client nodes. Our key generation process consists of generating MIN_i (client node

related), MIN_s (server related), $SRMP_{Ni}$ (based on screen resolution and mouse position) and T_i (which is a time component). Based on $SRMP_{Ni}$ and T_i, a random number of 128 bits R_i is generated. The MIN_i and MIN_s make use of MAC address, IP address, host name of the server S and node N_i respectively. So, the key comprises of various components and is a combination of various server and client related information. This makes it hard for the attacker to guess the key.

The step by step procedure is as follows:

KeyGeneration (n)

{

/* n represents the number of bits required to represent the NOB variable. */

/* acquiring N_i's information */

1. Acquire node N_i's MAC address (MAC_{Ni}), IP address (IP_{Ni}), Host Name (N_{Ni}), Screen Resolution (SR_{Ni}), Mouse Position (MP_{Ni}).

/*generate MIN_i of 128 bits*/

2. Generate MIN_i a variable by appending MAC_{Ni}, IP_{Ni}, N_{Ni}.
3. If MIN_i < 128 bits
   Pad '0' in the MIN_i
   goto step 3
4. If MIN_i > 128 bits
   Retaining the most significant 128 bits of MIN_i and discard the remaining bits.

/* acquiring server's information */

5. Acquire Server MAC Address (MAC_s), IP address (IP_s), ServerName (N_s).
6. Acquire current Date (dd/mm/yyyy), Time (hh:MM:ssss) from server when node N_i occurred.

/*generating MIN_s of 128 bits*/

7. Generate MIN_s a variable by appending MAC_s, IP_s and N_s.
8. If MIN_s < 128 bits
   Pad '0' in the MIN_s
   goto step 8
9. If MIN_s > 128 bits
   Retaining the most significant 128 bits of MIN_s and discard the remaining bits.

/*generate T_i of 128 bits*/

10. Generate T_i by appending dd, mm, yyyy, hh, MM and ssss in the string format.

/* generate random number R_i */

11. Generate 128 bits $SRMP_{Ni}$ a variable by appending 64 bits SR_{Ni} with 64 bits MP_{Ni}

/* SR_{Ni} will be in the string format (width appended by length). e. g. 10240768 where width= 1024 and length=768*/

/* MP_{Ni} will be in the string format (x-coordinate appended by y-coordinate). e. g. 07400568 where x-cord= 740 and y-cord=568*/

12. R_i=$SRMP_{Ni}$ (XOR) Ti /* Here, R_i is 128 bits */

/* calculating n bits NOB variable from R_i for dynamic key generation */

13. NOB = last n-7 bits of R_i.
   /*(i. e. NOB=$R_{i((128\text{-}n) \text{ to } 128)}$)*/
   /* Here, no. of bits in (NOB) = n-7 bits*/
14. append seven '0's in to the LSB side of NOB
   /* Here, no. of bits in (NOB) = n bits*/
15. If NOB<128
   NOB=128
   /* maximum value of (NOB) = (2^n-1), minimum value of (NOB) =128 where n is agreed value between the communicating parties. */

/* generating key K_i*/

/*MIN_{is} , $MINT_{is}$ are variables*/

16. MIN_{is} = MIN_i (XOR) MIN_s
17. $MINT_{is}$ = MIN_{is}(XOR) T_i
18. K_i = $MINT_{is}$ (XOR) R_i

/* dispatch key */

19. Dispatch key K_i and NOB to node N_i.
20. END

}

3.2 Encryption Algorithm

The encryption algorithm divides the whole message into 120 bits blocks and each block is appended by 8 bits message authentication code (MAC). Now, these 128 bits blocks are encrypted by the key. After every NOB bits of the message, the key is changed and encrypted by the new key. This key changing procedure is repeated till the end of the message. The step by step procedure is as follows:

Encryption (Key K, NOB, Plain Text)

{

1. Cipher_Text = null

/* At the end of ciphering process Cipher_Text variable will hold the whole cipher text of the whole plain text. */

2. M = First 120 bits of Plain Text

/* If plain text size is not of 120 bits. So, pad blank at the end of the message to make it of 120 bits.*/

3. bit_count = 0

4. If M = null

goto step 12

Else

goto step 5

5. 8 bits MAC = MD (NOB, M)

/*Here, we are generating 8-bits message authentication code by passing NOB as a key and a 120 bits message M into MD function of the communicating part's choice for maintaining message integrity.*/

6. 128 bits (M + MAC) = append 120 bits M with 8 bits MAC

7. 128 bits C = 128 bits (M + MAC) (XOR) 128 bits key K

8. Cipher_Text =Append C with Cipher_Text

9. bit_count = bit_count + 128

10. M= next 120 bits of Plain text /

* If plain text size is not of 120 bits. So, pad blank at the end of the message to make it of 120 bits. */

11. If bit_count != NOB

goto step 4

Else

{ /*new key generation*/

Perform shift operation at K

K = K + NOB

K = K (XOR) NOB

goto step 3

}

12. END

}

3.3 Decryption Algorithm

The decryption algorithm divides the whole cipher text into 128-bit blocks which are further decrypted by same multiple keys (which was used for encryption). But now, these keys are generated by decryption algorithm by taking only the initial key and NOB which was exchanged earlier. After NOB bit, the key is changed and the next blocks are decrypted by the respective new keys. 8-bit MAC and 120-bit message will be separated from the decrypted message. It calculates the MAC for 120-bit message and also performs the comparison between, the decrypted message MAC and the calculated MAC by decryption algorithm. If both are equal then message is integrated and proceeds for next block, otherwise the connection will be reset since the message has been altered by someone. After every NOB bits, the key will be changed and this procedure is repeated till the encrypted message ends. The step by step procedure is as follows:

Decryption (Key K, NOB, Cipher Text)

{

1. Plain_Text = null

/*At the end of deciphering process Plain_Text variable will hold the whole plain text of the whole cipher text*/

2. C = First 128 bits of Cipher Text

3. bit_count = 0

4. If C = null

goto step 11

Else

goto step 5

5. 128 bits (M + MAC) = C (XOR) 128 bits Key K

6. Separate 120 bits M and 8 bits MAC from 128 bits (M + MAC)

7. Append M with Plain_Text

8. C = next 128 bits of Cipher Text

9. 8 bits MAC' = MD (NOB, M)

/*Here, we are generating 8-bit MAC' by passing NOB as a key and 120 bits message M into MD function*/

10. If MAC' = MAC

{ /*Message integrity verified*/

bit_count = bit_count + 128

If bit_count != NOB

goto step 4

Else

{/*new key generation*/

Perform shift operation at k

K = K + NOB

K = K (XOR) NOB

Goto step 3

}

}

Else

/*Message not integrated*/

Select proper action (e. g. reset connection)

11. END

}

The maximum & minimum data which can be encrypted or decrypted by using same key is 2^n-1 bits & 2^7 bits respectively. We can vary the min and max boundary by modifying the require bit size to represent NOB which we have taken 'n'.

4. KEY BENEFITS IN THE PROPOSED ALGORITHM

1. The algorithm involves few XORs, SHIFTs, additions, comparisons and appends operations. So, the algorithm works faster and the run time complexity is less.

2. Due to the change of keys after the random bits, it is very hard to perform the cryptanalysis in order to deduce the secret keys.

3. Due to 128-bit key and n-bit NOB, the cipher becomes more secure. Because, a total $2^{128} + 2^n$ number of permutations are possible where 128 >= n>=7. So, brute force attack is much time taking, nearly $1.079x10^{28}$ year for a personal computer which permutes thousands of 128-bit numbers in 1 second for n=7. If we increase the value of n then the number of years required for brute force attack will increase. The lesser the size of n, the number of key generation is more. Hence, in both the cases, we are optimizing security.

4. Although the key is changing frequently but no need of key exchange. So, it reduces the network traffic.

5. If an attacker is so lucky and he does the best guess, the probability for guessing the key will be $(1/2^{128})$ or $2.938*10^{-39}$, for NOB it will be $(1/ 2^7)$ or $7.812x10^{-3}$ where n=7 and the joint probability for both will be $(1/2^{128})*(1/ 2^7)$ or $2.295*10^{-41}$. If we increase the value of n then the joint probability of best guess will decrease.

6. This mechanism achieves message confidentiality as well as integrity where as other mechanisms only provides message confidentiality.

7. The time requires to achieving message Confidentiality is comparably much less than other existing mechanisms.

5. SIMULATION RESULTS

We have used java cryptographic environment (JCE) as a coding standard. To perform the simulation, we have used a desktop with the following specification [9]-[11]. Windows XP sp2 as the operating system with java run time environment (JRE), Intel Core^{tm}2 Duo 2.53 GHz processor and 2 GB RAM. We have implemented our algorithm and compared it with DES, 3DES, AES and Blowfish. The time delays for encrypting/decrypting of 1Kilobit of data are as follows:

Table 1. Time taken for Encryption/Decryption

Algorithm	*Encryption Time (second/Kbit)*	*Decryption Time (second/Kbit)*
DES (56 bit key)	0.296957406	0.237437491
3DES (56bit three key)	0.765329835	0.709092870
AES (128 bit key)	0.264974528	0.265052825
Blowfish (128 bit key)	0.204341515	0.001350465
BCMKSE	0.177459598	0.000123706

(128 bit key)		

On the basis of table 1, we plotted the following Graphs. The Graphs are plotted for the algorithms against their respective times taken for encryption/decryption.

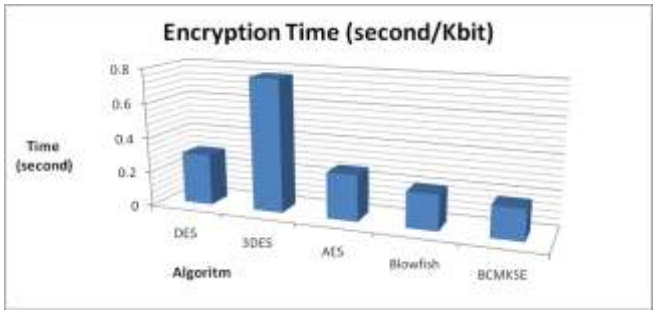

Figure 1. Time taken for Encyption

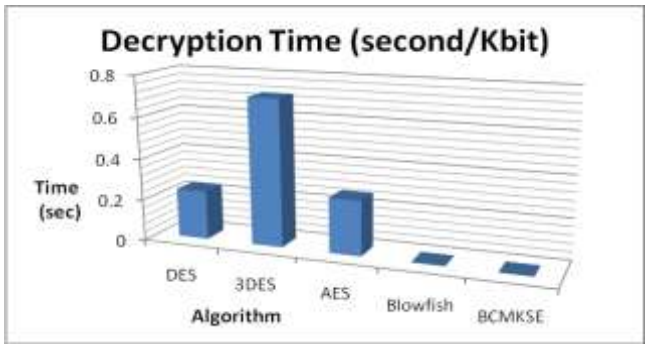

Figure 2. Time taken for decyption

So, considering the evidences provided by the above stated algorithm, it is clear that the time required to encrypt/decrypt 1Kbit data taken by BCMKSE algorithm is lesser than the existing symmetric key algorithms. Additionally BCMKSE algorithm also provides message integrity. The following are the snapshots of Key and NOB generation, Encryption and Decryption Operations for 1 kilobit data where number of bit require to represent NOB is 16.

Figure 3. Encryption with BCMKSE

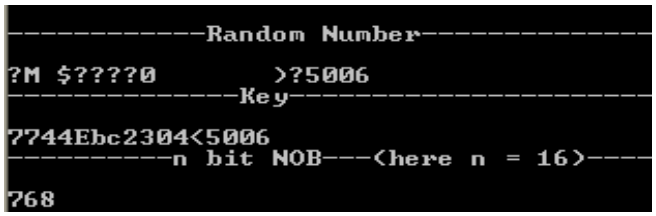

Figure 4. Key and NOB generation

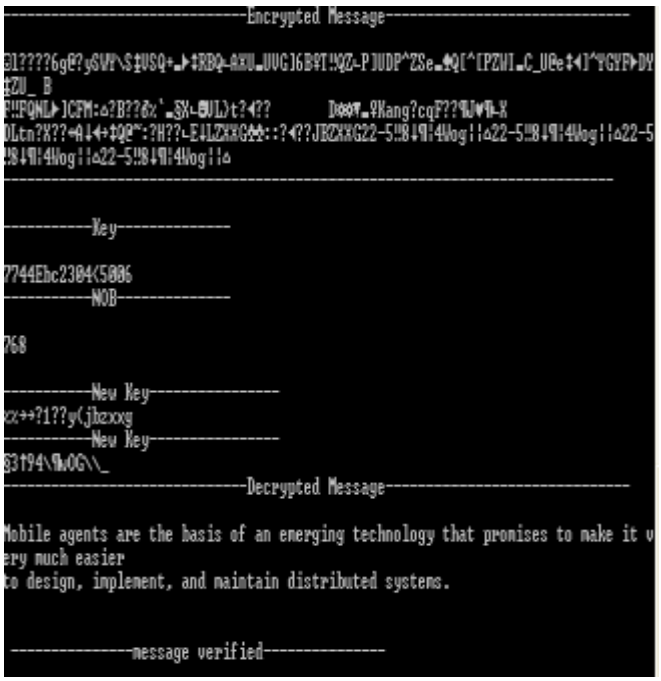

Figure 5. Decryption with BCMKSE

The proposed algorithm shows how multiple keys have been used to encrypt the message and these multiple keys are being decided by the NOB variable. So, the NOB and key will be different at any time not only for all the nodes, but also for the individual node and for the same message as well. In this experiment NOB equals to 768 or $6*2^7$ (that means after every 96 characters of encryption/decryption, the key will be changed).

6. CONCLUSION

In this paper, algorithms for key generation, encryption and decryption are proposed. The proposed algorithms are efficient because they are simple and easily implementable. The algorithm achieves message confidentiality with less run time complexity and computational overhead as compared to the existing algorithms, which has been justified in the previous section. Also it provides message integrity which is not provided by other symmetric key algorithms. This paper presents a new way of using multiple keys concept without increasing message overhead. We can improve its efficiency by changing its implementation platform, language, programming style and network technology. Instead of developing a complex algorithm by involving complex and time taking operations, this paper emphasize to think about logically complex algorithm with simple operations.

7. REFERENCES

[1] Wikipedia Symmetric key algorithm. Updated February, 2007. http://en.wikipedia.org/wiki/Symmetric_key_algorithm (March, 2009).

[2] H W. Stallings. *Cryptography and Network Security: Principles and Practice*, 3rd Edition. Prentice Hall, New Jersey, USA, 2003.

[3] FIPS Publication 46-3, "Data Encryption Standard (DES)." U.S. DoC/NIST, October 25, 1999.

[4] American National Standard for Financial Services X9.52-1998, "Triple Data Encryption Algorithm Modes of Operation." American Bankers Association, Washington, D.C., July 29, 1998.

[5] FIPS Publication 197, "Advanced Encryption Standard (AES)." U.S. DoC/NIST, November 26, 2001.

[6] Wikipedia Advanced Encryption Standard. Updated February, 2007. http://en.wikipedia.org/wiki/Advanced_Encryption_Standard (March, 2009).

[7] B. Schneier, Description of a New Variable-Length Key, 64-Bit Block Cipher (Blowfish) Fast Software Encryption, Cambridge Security Workshop Proceedings (December 1993), Springer-Verlag, 1994.

[8] B. Schneier, Applied Cryptography, John Wiley & Sons, New York, 1994.

[9] Menezes, P. van Oorschot, and S. Vanstone, "Handbook of Applied Cryptography." CRC Press, New York, 1997.

[10] GridCrypt: High Performance Symmetric Key Cryptography using Enterprise Grids by Agus Setiawan, David Adiutama, Julius Liman, Akshay Luther and Rajkumar Buyya Grid

Computing and Distributed Systems Laboratory Dept. of Computer Science and Software Engineering The University of Melbourne, Australia.

[11] Performance of the AES Candidate Algorithms in Java by Andreas Sterbenz, Peter Lipp Institute for Applied Information Processing and Communications Graz, University of Technology Inffeldgasse 16, A-8010 Graz, Austria.